AI Motor Evaluation Kit
User's Manual

The AI Motor Evaluation Kit was designed to be a complete low-cost way to get hands-on experience with an AI Motor. Designed specifically for robotics, AI Motors (AIMs) are building blocks combining mechanical connectors, servo-motor actuator, sensors, and smarts. All motors communicate via a single serial port, but at logic rather than RS232 voltage levels. Unique features include programmable servo and full-rotation modes, position feedback (position encoder and turns counter), and mechanical loading feedback.

This kit combines an AI-701 motor with an RS232 level converter board, wall-wart power supply, serial PC cable, and example software. With the included software GUI you can initialize and manually control motors and tune proportional-differential servo control coefficients. We include example C code for controlling AI Motors (AIMs) from an embedded microprocessor. In the coming months we'll be developing example VisualBasic and VisualC code, and making it available by free download. If you have at least 2 motors, you can run the Pan-Tilt demo code and have the same functionality as the MGR-1002 kit. The pan-tilt demo allows you to record mechanically moving the motors by hand, and then play back the motion sequence. The RS232 level converter includes a header of test-points, allowing you to use it in other serial testing and prototyping as well. The included power supply supports up to 5 AIMs at typical loading.

This manual is divided into two sections. The first, Quick Start, lists the kit contents then walks you through the examples using the AI Motors Tool and Pan-Tilt Demo PC software. The second section summarizes specifications for AI Motors and provides specifications for the converter board.

Part 1: Quick Start

This quick start walks you through initializing and controlling AI Motors through the AI Motor Tool GUI and using the Pan-Tilt Demo. For additional information on AI Motors, see the AI-701 manual. More information on the converter board is included later in this document.

To use the software, you'll need a personal computer running Windows. WinXP is preferred; Win3.1/NT/95/98/2000 should be fine.

1.1 Kit Contents

- 1 AI-701 motor (7 kg*cm stall torque) with
  - 11 mechanical attachment pieces plus screws and nuts
  - 2 cables
  - replacement gears
- RS232 level-converter printed-circuit board
- 1 amp. 7.5 V unregulated DC power supply
- Serial cable (DB9 F to RJ11-4 adapter and RJ11-4 cable)
- CD-ROM containing
  - AI Motor Tool
  - Pan-Tilt GUI
  - AI-701 manual
  - This document

NOTE: *Don't substitute an unregulated supply rated at higher than 7.5 volts*. Most DC supplies are unregulated; when lightly loaded their actual output voltage may exceed the AIM specified maximum. The 7.5 V 1 A supply with the light loading of a single motor results in a roughly 10 V output. A 7.5 V 1 A supply is adequate for 1—5 AI Motors.

1.2 Using the AI Motor Tool

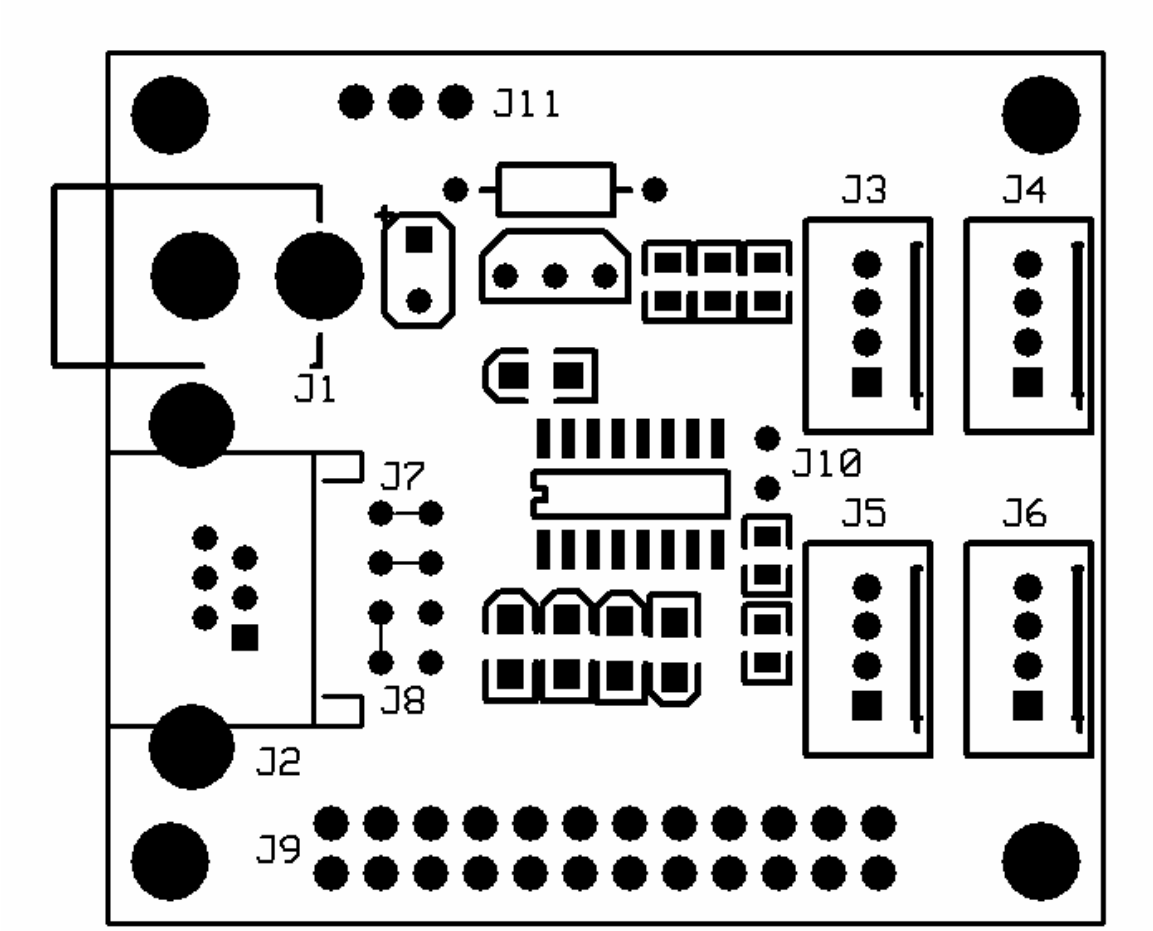
The AI Motor Tool, provided by the motor manufacturer, is a graphical user interface used to initialize and manually control AI Motors. It isn't fully implemented, so we'll walk you around the blind spots.

1.2.1 Installing the AI Motor Tool

The AI Motor Tool is included on the kit's CD-ROM, and can be downloaded from www.Garage-Technologies.com. There's no installation process; just go to the CD-ROM and copy AI Motor Tool to wherever you'd like to keep it on your PC.

1.2.2 Example 1: Initializing the Motors

This example walks you step by step on how to use the GUI to initialize the motors.



Errata: silkscreen shows voltage regulator reversed from actual and proper installation.

Figure 1 – Connectors and Jumpers

1. Connect a the serial adapter cable between your PC's serial port and the RS232 converter board **J2**. Click on the icon to run the AI Motor Tool. Be sure the *SingleSetting* tab is selected.
2. The AI Motor Tool communicates via the PC's serial port. For most PCs this is Com 1, so select Com1 in the upper left of the GUI. AI Motors ship with a default ID of 0 and a default baud rate of 57,600, so set the tool's baud rate to 57600 (also in the upper left of the GUI).
3. Connect only one AI Motor to the converter board. Any of the 4 connectors **J3—J6** can be used – they're all the same serial buss.
4. Plug the power adapter into a wall plug and then into the converter board **J1**. A small green LED will turn on.

5. To establish communications, click on the *Baud Scan* button. 57600 should now be displayed above it. Click on the *ID Scan* button. Scan ID should now show 0. Both are fine for the first motor.
6. Assuming you haven't changed the *Act ID* and *Act Mode* (on the left side of the GUI) from 0, wiggling the slider under the *Command Button* will cause to motor to move.
7. You can also set the ID of this motor to 1 (or any other ID up to 31). Select 1 from the *Set ID* menu, then click on the *ID Set* button.
8. After selecting 1 for the Act ID, you can now control the motor from the slider.

It's important to have only one motor connected when first initializing a motor and giving it an ID number. All motors are shipped with the a default ID of 0, and having more than one motor with the same ID connected can disrupt communications in a very confusing way. Once you've given each motor a unique ID number, you can connect up to 31 motors to a single serial buss and manually control them with the AI Motor Tool.

The SingleSetting panel also allows you to examine and change the various settings for the AI Motors. To change a setting, the *ID Scan* must agree with the ID of the motor you want to initialize. To change the baud rate, first use *Baud Set* and after that change the GUI's baud rate (upper left corner).

The following examples assume that the motor is set to ID 0, with other settings at the factory default.

1.2.3 Example 2: Manual Control

The AI Motor Tool can be used to issue single commands from the leftmost part of the GUI screen. These commands are as follows.

- Act ID – selects which motor ID to command.
- Act Mode
    - 0-4 sets the speed. The numbering is reversed (0 is fastest, 4 is slowest)
    - 5, PositionRead, issues a command for the motor to return its current position (and load)
    - 6, "MotorDown", turns off the motor but leaves the motor's microprocessor active. This command should be issued when you want to manually move the motor to a particular position (otherwise the motor will fight to maintain the last instructed position).
    - 7, PowerDown, removes torque from the motor (as in Command 6) and puts the motor's microprocessor in power-conserving standby. It comes out of standby at the next command.
    - 8, "WheelActCCW", puts the motor in full rotation mode moving counter-clockwise.
    - 9, "WheelACTCW", puts the motor in full rotation mode moving clockwise.

NOTE: The Position Read mode also turns off motor torque. To get position and loading feedback without affecting the motor, just re-issue the last command.

The following walks you step by step through commanding a motor to a specified position, looking at motor current loading, reading the position of a motor, and controlling full-rotation.

1. Connect a motor to the converter board. This can be either directly to any of the connectors in Figure 1, or by daisy-chaining the motors.
2. At the far left of the AI Motor Tool GUI, use *Act ID* to that of the motor. Select *Act Mode* of 0 to 4 for whatever speed you'd like.
3. Enter a value from 0 to 255 into *Command Value*. This is the position that will be commanded.
4. Click the *Command Send* button to issue the command. You can also use the slider to send out a rapid series of position commands. Experiment with a few positions and speeds.
5. You'll notice that each commands results in two additional values being displayed in *Return Value* at the bottom of the GUI. These values are in hexadecimal. For most commands, the first of each pair indicates motor current[1] and the second is the position. Both values are at the time the command was issued. So current will be 0 or very low, since the motor isn't doing any work while waiting for the command, and the position will be the prior target position. Add one of the mechanical attachments (like the geared wheel) to the side axel and apply some resistance while giving another position command. You'll see the current value go up. And that 7 kg*cm is pretty strong.
6. Select *Act Mode* of 6 and click *Command Send* to remove torque from one of the motors. Mechanically move the motor to a new position, then select *Act Mode* of 5 and click *Command Send*. The new position will be displayed in the *Return Value* panel (the Command Value won't be updated).
7. Select *Act Mode* of 7, select a *WheelSpdLevel* (here a higher number is faster), then click *Command Send*. The motor will rotate counter-clockwise until it's given a new command. If that command is another CCW or CCW rotation, the Return Values will be the number of rotations since the last command (rolling over after 255 decimal) and the current position.

Just remove the power supply plug from the converter board jack to turn everything off.

1.2.4 Using the "ActionCon" Panel

The ActionCon and GraphView panels are more than a bit confusing – we're still figuring them out ourselves. But together, you can use them to visualize AI Motor performance and fine-tune the proportional and differential coefficients.

Most closed-loop control such as servo motors uses the "PID Algorithm" (Proportional, Integral, and Differential) to provide rapid and stable control. With the right coefficients

[1] Multiplying the current value, converted to decimal, times 18.5 gives the approximate motor draw in milliamps. Current draw is roughly proportional to the mechanical loading on the motor.

for each term, the motor will quickly go to the correct speed or position, without overshoot or oscillation (hunting). Get the coefficients way-wrong and the motor goes berserkers. AIMs come with default settings that provide good performance for nearly all conditions. And their built-in microprocessor even tweaks the coefficients for different supply voltages. But you can still adjust the proportional and differential terms yourself. The integral term is primarily useful in maintaining a given speed, so many robotic applications involving positioning drop this term.

The *Action Control Panel* allows you to set positions for all the motors at once. It does *not* allow you to enter and sequence through a series of set positions. In the spread-sheet like section you can enter target position for each motor, then send a command to all those motors.

1 Clicking on the ScanID tab will automatically scan all the motors connected, and list their ID and current position.
2 You can enter a target position for each motor by editing the table. If you enter a number above 255 the position will roll-over past 256 (entering 300 is the same as entering 44). Pressing the TarPosALL tab will copy the first target position into all the motors.
3 Clicking on "ActionRun" will send the appropriate commands, and collect position and load information during and after motor operation.

We're still trying to figure out the Period, SampleNum, and Increment columns. Entering values in the Increment column will cause the motor to slowly change position during "ActionRun." Clicking the various "…All" buttons will copy the first row's entry into all the later rows. And "Synchronized Position Send" appears to send a position command to all the motors, but without sampling the data needed for the GraphView panel. You need to set the *LastID* if you have more than one motor to operate.

1.2.5 Using the "GraphView" Panel

After using the Action Control Panel, you can go to the Graph View Panel to display a plot of motor performance.

1. Select the motor ID and data to plot from the drop-down boxes. You can select up to 6 lines to be plotted, and each line can be the reference (target) position, current position, current torque (motor load), or current speed for a given motor ID. The defaults are fine for a motor set to ID 0.
2. Un-click the ScaleChecked box to see the raw data units.
3. Go back to the Action Control Panel, enter a new TarPosition, and click the ActionRun button.
4. Go back to the Graph View Panel and click the *ViewGraph* button to plot the data.

1.3 Using the Pan-Tilt Demo

If you have 2 or more motors, you can also try out the pan-tilt demo. It's pretty cool. You *must* have 2 motors for this software to work – it waits for data to be returned from motor IDs 0 and 1.

1.3.1 Assemble and Initialize the Motors

1. Use the AI Motor tool to set one motor to ID 0 and another to ID 1, as described above. You'll also need to change the baud rate from the default 57,600 baud. Use *SetBaud* to set each motor to 115,200. Once the motor has its new rate, you'll need to change the command baud rate in the upper left of the Command Setting Panel.
2. Place the motors into some useful or interesting geometry. For tilt-pan action,
    i. Place the motor ID 0 with the 4 mounting brackets on the bottom and the shaft vertical. This is the pan motor. You may want to attach it to a board with some very small wood screws.
    ii. Use a shaft-to-housing connector piece to attach the second motor (ID 1). The second motor is now the tilt motor, with a horizontal shaft
    iii. Attach some connector piece to the second motor's shaft to make it easier to move that motor by hand.
3. Plug both motors into the converter board… or plug one motor into the converter board and the second motor into the first motor.

1.3.2 Install the Pan-Tilt Demo Software

The Pan-Tilt Demo software is included on the kit's CD-ROM, and can be downloaded from www.Garage-Technologies.com. There's no installation process; just go to the CD-ROM and copy Pan-Tilt Demo to wherever you'd like to keep it on your PC. Click on the icon to run it.

1.3.3 The Demo

You're now running the pan-tilt demo. The left and right buttons rotate the pan motor counter clockwise and clockwise. The up and down buttons rotate the tilt motor up and down. The other buttons take a bit more explaining.

This demo uses the PositionRead command described in the second part of this document to periodically record the positions of both motors as you physically move them around with your hands (*not* the up, down, left, and right buttons). This sequence of positions is stored, and can be used to automatically repeat the motion sequence.

1. Click the *REC* button to start recording a movement sequence.
2. Physically move the motors around by hand.

3. *Click anywhere else on the GUI to stop recording*. If not, the program will fill its buffer then stop running after a few seconds.
4. Now, pressing PLAY will playback the sequence.

The *USER PROGRAM* button links to a routine in the code. You'd need to edit the source code and re-compile to use this feature.

Part 2: Specifications

This section summarizes specifications for the AI Motors, and provides the full specifications for the converter board.

2.1 AI Motors

This evaluation kit comes with one AI-701 AI Motor. For the full specifications, features, and description of use, see *The AI Motor-701 Manual*. The '701 and '1001 motors are interchangeable, the primary difference being that the '1001 uses a higher gear ratio to achieve higher torque at lower speed.

2.1.1 Summary of Specifications

- Stall torque 7 kg*cm @ 9.5 V
- Maximum speed 82 rpm @ 9.5 V
- Weight 40 g
- Size 51.6 x 34.3 x 37.1 mm
- Power
    - Maximum 650 mA @ 5 VDC, 1000 mA @ 10 VDC
    - Typical 100—200 mA
    - Minimum 12 mA
- Positioning range and resolution
    - Low resolution mode – 0 to 333.3 degrees, 1.31 degrees resolution
    - High resolution mode – 0 to 166.6 degrees, 0.65 degrees resolution
- Speed control
    - Position mode – 16 speeds
    - Full rotation mode – 5 speeds
- Up to 31 motors on a single serial bus (asynchronous serial at TTL levels)
- Baud rate 2,400 to 460,800 bps
- Other settable parameters
    - Movement bounds
    - Over-current threshold
    - P- and D-gain

2.1.2 Summary of Protocol

The following table summarizes the AI Motor protocol. The most used command packets are 4 bytes long. Initialization commands are 6 bytes long. All command packets except synchronous move are acknowledged by 2 bytes from the motor. Here we use conventional C notation for packing data into the command bytes. For some commands certain bit-fields in command bytes 3, 4, and 5 can have arbitrary values, but that is ignored in this simplified table. A blank in the table indicates that no byte is sent (4-byte commands).

There is a special command protocol to move several motors simultaneously. The motor IDs must be sequential and start at 0.

- Cmd Byte1 – 0xFF
- Cmd Byte2 – (spd) << 5 | 0x1F
- Cmd Byte3 – last ID
- Cmd Byte4 – position for ID=0
- Cmd Byte5 – position for ID=1
- …
- Cmd ByteN – position for last ID
- Cmd ByteN+1 – checksum = exclusive-or of all positions anded with 0x7F

There are no acknowledgement bytes for the synchronous position send command.

Name	Cmd Byte1	Cmd Byte2	Cmd Byte3	Cmd Byte4	Cmd Byte5	Cmd Byte6	Ack Byte1	Ack Byte2
Position Send	0xFF	(spd[2])<<5 | ID	Position	Chksum1[3]			Current	Position
Position Read	0xFF	0xA0 | ID	0x00	Chksum1			Current	Position
Act Down[4]	0xFF	0xC0 | ID	0x10	Chksum1			Cmd Byte3	Position
Power Down	0xFF	0xDF	0x20	Chksum1			arbitrary	arbitrary
Full Rotation	0xFF	0xC0 | ID	(dir[5]) << 4 | (spr[6])	Chksum1			Net rotation count	Position
Synch-ronous Move	See text	See text	See text	See text	See text	See text	None	None
Set Baud	0xFF	0xE0 | ID	0x08	rate[7]	Cmd Byte4	Chksum2[8]	New baud	New baud
Set Gains	0xFF	0xE0 | ID	0x09	Pgain	Dgain	Chksum2	New Pgain	New Dgain
Set ID	0xFF	0xE0 | ID	0x0A	New ID	New ID	Chksum2	New ID	New ID
Read Gains	0xFF	0xE0 | ID	0x0C	0x00	0x00	Chksum2	Pgain	Dgain
Set Resolution	0xFF	0xE0 | ID	0x0D	Resol, 0 or 1	Cmd Byte4	Chksum2	New resol	New resol
Read Resolution	0xFF	0xE0 | ID	0x0E	0x00	0x00	Chksum2	0/1 (resol)	0/1 (resol)
Set over-current threshold	0xFF	0xE0 | ID	0x0F	Threshold mA/18.5	Cmd Byte4	Chksum2	New Thresh.	New Thresh.
Read over-current threshold	0xFF	0xE0 | ID	0x10	0x00	0x00	Chksum2	Thresh.	Thresh.
Set Bounds	0xFF	0xE0 | ID	0x11	Min pos	Max pos	Chksum2	New min pos	New max pos
Read Bounds	0xFF	0xE0 | ID	0x12	0x00	0x00	Chksum2	Min pos	Max pos

Table 1 – AI Motor Commands

2.2 Converter Board

The AI Motor Evaluation kit includes an assembled printed circuit board combining RS232—TTL level converter, power connector and regulator, and AIM connectors.

[2] "spd" is 0 to 4 indicating speed; 0 is fastest
[3] Checksum1 = Byte2 ^ Byte3 & 0x7F
[4] Turn off motor torque.
[5] "dir" is direction, 3=counter clockwise, 4=clockwise.
[6] "spr" 0 to 15 indicating rotation speed; 0 is stop, 15 is fastest
[7] See AI-701 User's manual
[8] Chksum2 = (Byte2 ^ Byte3 ^ Byte4 ^ Byte5) & 0x7F

2.2.1 Mechanical

Figure 1 shows the connectors, jumpers, and mounting hole locations. Outer dimensions are 1.75 x 2.00 inches.

2.2.2 Power

External power is provided by battery or UL listed power converter. Input power must be 6.5 to 10 V DC. Input voltage is directly routed to the AI Motor connectors, and regulated to 5 V DC for on-board circuitry. Current draw is approximately 10 mA, excluding the motors. The power connector is a 2.1 mm jack, center positive (**J1**).

2.2.3 Other External Connections

The converter board provides the following additional external connectors.

- **RJ11-6P4C** female connector for true RS232 voltage levels (**J2**).
- **4 2mm headers** for AI Motors **(J3—J6).**
- **Pads for 2x12 pin .1" header (J9)**
- **Pads for raw power access (J11)**

As shipped, the serial adapter cable provides the following connections:

DB9 (female)	**RJ11-6P4C (male)**
1	
2 RXD	2 (data from AIM to PC) (red)
3 TXD	3 (data from PC to AIM) (green)
4	
5 GND	1 (black)
6	
7	
8	
9	

Pin 4 (yellow) of the RJ11-6P4C connector is not connected. With the jumper changes described later, it can be used to receive a handshaking signal from the host.

Pinout for the AIM connectors is as follows:

1 Ground
2 AIM RX (data from PC to AIM)
3 AIM TX (data from AIM to PC)
4 Supply voltage

All AIM connectors are the same serial buss. Multiple AIMs can be daisy chained from a single connector; 4 connectors are provided for convenience.

The 2 x 12 array of .1" spaced pads can be used for a header connector or as a bread-boarding area. Corresponding pairs of pins on the two rows are connected together (see schematic in the Appendix). Pinout of the columns is as follows:

1 RJ11 (J2) pin 4 (normally grounded by J8)
2 RJ11 (J2) pin 3
3 RJ11 (J2) pin 2
4 +5 V regulated (low current; regulator is 78L05 in TO-92 package *without heat sink*)
5 MAX232 pin 7 (RS232 output, corresponding to J9 pin 7 TTL input)
6 MAX232 pin 8 (RS232 input, may be jumpered to J2 Pin 4, or ground; corresponds to J9 pin 8 RS232 output)
7 MAX232 pin 10 (TTL input, corresponding to J9 pin 5 RS232 output)
8 MAX232 pin 9 (RS232 output, corresponding to J9 pin 6 TTL input; may be jumpered to LED)
9 Ground
10 AI Motor TX (TTL-level data from AIMs; goes to MAX232 pin 11)
11 AI Motor RX (TTL-level data to AIMs; goes to MAX232 pin 12)
12 Raw supply power

Maximum external +5 V usage (pad column 4) is about 50 milliamps under typical conditions (office ambient temperature, 10 V unregulated input).

"J11" provides pads for accessing power.

- Ground
- Raw power (also direct to AI Motors)
- Raw power after LC filter

2.2.4 On-Board Jumpers

As shipped, the board has no jumpers, but there are pads that can be used for jumpering (see schematic in the Appendix). They are:

- J10 – adding a jumper here will connect the MAX232 RS232 output #2 (and J9 pin 8) to an LED with current limiting resistor. Nice for testing.
- J7 – cutting the traces on the back side of the board, you can use jumpering to swap pins 2 and 3 on the serial port connection. Though its probably easier to substitute a swapped RJ11 cable.
- J8 – cutting the trace on the back side of the board, you can jumper pin 4 of the serial connector to the MAX232 RS232 input #2. Modifying the DB9 adapter, this gives you access to a handshaking line.

2.2.5 Software Included

- Megarobotics' AI Motor Tool
- Megarobotics' Pan-Tilt Demo
- Open-source C example code (for AVR-GCC)

Appendix 1 – Example Code

The following code is a slightly reduced version of the demo code provided with another kit, the AIM Introductory Kit. This version shows the AI Motor functions, with fewer comments, and omits the fixed-path demo. Note that a "standard library" and initialization for the AI Motor running commands is less than 2 pages of code and each demo is less than 1 page.

```
//*********************************************************************
// File Name    : AIMintrolite.c
//
// Shortented version of AIMintro.c as program listing example
// of using AIM motors and the AIM Intro Kit
// GPL
// 17-Jan-2005 Garage Technologies, Inc.
//*********************************************************************

//----- Include Files ---------------------------------------------------
#include <avr/io.h>             // include I/O definitions (port names, pin names, etc)
#include <avr/signal.h>         // include "signal" names (interrupt names)
#include <avr/interrupt.h>      // include interrupt support

#include "global.h"             // include our global settings
#include "uart.h"               // include uart function library
#include "rprintf.h"            // include printf function library
#include "timer.h"              // include timer function library (timing, PWM, etc)
#include "a2d.h"                // include A/D converter function library

#define abs(A)(((A)>0) ? (A) : (0-(A)))

//*****************************AIM "Library"*******************************
// should be enums...
#define aPOS        1       // AIM send position
#define aPOSr       2       // AIM read  position
#define aDWN        3       // AIM act-down (remove torque)
#define aPDN        4       // AIM power-down
#define aROT        5       // AIM full-rotation
#define CCW         3       // AIM full-rotation direction
#define CW          4       // AIM full-rotation direction
#define LEFT        1       // left AIM
#define RIGHT       0       // right AIM

//transmit 4-byte AIM command
void aimcmd4(u08 b2, u08 b3)
{
        rprintfChar(0xFF);
```

```
        rprintfChar(b2);
        rprintfChar(b3);
        rprintfChar((b2^b3)&0x7F);
}

// get  from AI motor
// NOT CHECKING FOR UART BUFFER EMPTY
u16 aimAck(void)
{
        u08 b1=0,b2=0;

        timerPause(1); // give motor time to respond
        uartReceiveByte(&b1);
        uartReceiveByte(&b2);
        return((b1<<8) | b2);
}


u08 hiByte(u16 x)
{
        return (x>>8);
}

u08 loByte(u16 x)
{
        return (x&0x00FF);
}

// send a command to AI motor
u16 aimCmd(u08 cmd, u08 id, u08 pos, u08 spd)
{
        u16 readback = 0x0000;

        readback = 0x0000;
        switch(cmd)
        {
                case aPOS:      // send a positioning command
                        aimcmd4((spd<<5)|id,pos);
                        readback = aimAck();
                        break;
                case aPOSr:     // command read-position
                        aimcmd4(0xA0|id,0x00);
                        readback = aimAck();
                        break;
                case aDWN:    // act-down
                        aimcmd4(0xC0|id,0x10);
```

```
                readback = aimAck();
                break;
        case aPDN:     // power-down
                aimcmd4(0xDF, 0x20);
                readback = aimAck();
                break;
        case aROT:     // full rotation modes
                aimcmd4(0xC0|id, (pos<<4)|spd);
                readback = aimAck();
                break;
        default:
                break;
    }

    return readback;
}

//***** Begin Code ************************************************************
int main(void)
{
    u08 i=0;          // unsigned 8 bit
    u08 curr,pos;
    u16 myack;
    u08 leftLight, rightLight;

    // ----- Initialization ---------------------------------------------

    // initialize our libraries
    // initialize the UART (serial port)
    uartInit();
    // make all rprintf statements use uart for output until told otherwise
    rprintfInit(uartSendByte);
    uartSetBaudRate(57600);       // AI motor default
    // initialize the timer system
    timerInit();
    // turn on and initialize A/D converter
    a2dInit();
    // Make the analog port input, be sure pull-ups are turned off
    DDRA = 0x00;
    PORTA = 0x00;
    // set a2d prescaler (speed and accuracy) and select voltage reference
    a2dSetPrescaler(ADC_PRESCALE_DIV32);
    a2dSetReference(ADC_REFERENCE_AVCC);
```

```
// ------------------- position & force readout demo -------------------
        // move both motors to middle
        aimCmd(aPOS,LEFT,128,1);
        aimCmd(aPOS,RIGHT,128,1);
        timerPause(1000);       // wait a second
        // remove torque from left motor
        aimCmd(aDWN,LEFT,0,0);
        curr = 0;
        // left motor as sensor
        while(curr<3)  // while right motor not loaded
        {
                // get position of left motor
                myack = aimCmd(aPOSr,LEFT,0,0); // also removes torque
                pos = loByte(myack);
                //send position to right motor
                aimCmd(aPOS,RIGHT,pos,1);
                timerPause(150); // let left motor get to position (with reduced current)
                // get current of right motor
                myack = aimCmd(aPOS,RIGHT,pos,1);
                curr = hiByte(myack);
        }
        // right motor as sensor
        for(i=0;i<50;i++)
        {
                // get position of right motor
                myack = aimCmd(aPOSr,RIGHT,0,0);
                pos = loByte(myack);
                // send position to left motor
                aimCmd(aPOS,LEFT,pos,1);
                timerPause(100);
        }


        // ------------------ shut-down motors -------------------
        aimCmd(aPDN,0,0,0);


        return 0;
}
```

Appendix 2 – Schematic